ETHICAL HACKING MEMORY CORRUPTION (EHMEC)

DESCRIÇÃO

(Nível Básico)

Carga Horária: 40 hrs

O curso Ethical Hacking Memory Corruption (**EHMEC**) apresenta conceitos fundamentais sobre vulnerabilidades e *exploits* relacionados à corrupção de memória em sistemas Linux e Windows.

Diariamente, novas falhas são descobertas em diversos sistemas, seja ele uma aplicação web, um *software* customizado, um componente do sistema operacional, que esteja instalado numa estação de trabalho, servidor ou dispositivo móvel. Durante muito tempo e ainda nos dias atuais, é possível se deparar com programas ou executáveis que foram concebidos com fraquezas em seu processo desenvolvimento.

Nesse treinamento, serão trabalhadas as técnicas básicas de criação de *exploits* para a exploração de binários nas plataformas de 32 bits (x86) e 64 bits (x86-64), tanto para o Sistema Operacional Linux quanto para o Windows, com foco nas vulnerabilidades de corrupção de memória e B*uffer Overflow* baseado na pilha.

Diversos temas serão abordados, tais como: arquitetura de computadores, gerenciamento de memória, registradores, estrutura dos binários, linguagem Assembly x86, utilização de *debuggers* para Linux e Windows, organização e funcionamento da pilha, Shellcode, como detectar e depurar uma falha de corrupção de memória, entender e explorar vulnerabilidades de *buffer overflow*, proteções existentes, técnicas básicas de *bypass* de proteção, fundamentos de criação de *exploits*, entre outros.

O **EHMEC** prepara o aluno para cursos mais avançados da GoHacking como o Ethical Hacking Exploit Development (**EHXD**) e o Ethical Hacking Reverse Engineering Malware (**EHREM**).

MÓDULO 0x00

1. Fundamentos de Arquitetura de Computadores
   - Bits, Bytes, Hexadecimal
   - Breve Histórico
   - x86 – 32 bits
   - x64 – 64 bits
2. Fundamentos de Gerenciamento de Memória
   - Tipos de Memória
   - Modo de Operação da CPU
3. Registradores
   - Tipos (32 e 64 bits)
   - Nomenclatura
   - Finalidade
4. Estrutura de Binários
   - Object File
   - *Linkers/Loaders*
   - ELF
   - PE
   - Carregamento em memória

MÓDULO 0x01

1. Introdução à Linguagem Assembly x86
   - Sintaxes Intel e AT&T
   - Mnemônicos e OpCodes
   - Comandos Básicos
2. Introdução ao GNU *Debugger* (GDB)
   - Principais funcionalidades
   - Principais comandos
   - Ferramenta Objdump
   - Comandos ltrace e strace
   - **Laboratório**
3. Organização e Funcionamento da Pilha
   - Chamada de Função
   - Estrutura da Pilha
   - *Endianness*
4. Introdução ao Shellcode em Linux
   - Fundamentos – *System Calls*
   - Compilação de um Shellcode pré-criado
   - Criação automatizada de Shellcode com Metasploit
   - Criação manual de Shellcode para Linux
   - **Laboratório**

MÓDULO 0x02

1. Corrupção de Memória
   - Conceitos e Fundamentos
   - Fuzzing e Metodologias
2. Entendendo o mecanismo do *Buffer Overflow* (BOF)
   - BOF Clássico – Vanilla
   - BOF baseado na Pilha
   - Controlando o EIP e o fluxo de execução de um programa
3. Introdução ao *Debugger* GEF
   - Instalação e configuração
   - Principais funcionalidades
   - Principais comandos
4. *Buffer Overflow* em Linux (x86)
   - Exploração básica de um binário Linux (ELF)
   - Procedimentos para detecção e exploração de BOF
   - BOF Vanilla em Linux, na prática, utilizando o GEF
   - Identificação de ponto de entrada vulnerável
   - Análise da corrupção de memória
   - Controlando o EIP e o fluxo de execução do binário
   - Saltando para endereços de memória de interesse
   - Entendendo e utilizando NOP
   - Criação de *Exploit* para execução do Shellcode
   - **Laboratório**
5. Proteções e Controles
   - NX/DEP
   - ASLR
   - PIE
   - *Canaries/Stack Cookies*
   - RELRO
   - *Fortify*
   - RPATH
6. Bypass da Proteção NX utilizando a técnica ret2libc
   - Limitações e Mecanismos de Contorno
   - Exploração de um binário Linux (ELF) com NX ativado
   - Emprego da Técnica ret2libc, na prática, utilizando o GEF
   - Criação de *Exploit*
   - **Laboratório**
7. *Buffer Overflow* em Linux (x64)
   - Análise de um binário Linux (ELF) de 64 bits
   - Principais diferenças de BOF em x86 e x86_64
   - BOF x64 Linux, na prática, utilizando o GEF
   - Criação de *Exploit*
   - **Laboratório**

MÓDULO 0x03

1. Corrupção de Memória em Sistemas Windows
   - Conceitos e Fundamentos
2. Introdução a *Debuggers* para Windows
   - WinDBG
   - x32dbg / x64dbg
   - Immunity Debugger
   - Aumento de funcionalidades do Immunity com Mona.py
   - Análise de um binário Windows (PE)
   - **Laboratório**
3. *Buffer Overflow* em Windows (x86 – 32 bits) – Programa 1
   - Análise de uma aplicação/programa real
   - Identificação de ponto de entrada vulnerável
   - Criação de uma Prova de Conceito (PoC) com Python
   - Análise de corrupção de memória da aplicação via Immunity Debugger
   - Como sobrescrever o endereço de retorno de forma precisa via BOF
   - Controlando o EIP e o fluxo de execução da aplicação
   - Identificando endereços de memória e instruções de interesse
   - Criação de Payloads
   - Entendendo, identificando e evitando BadChar
   - Entendendo e utilizando NOPs
   - Criação do *Exploit* e execução de Shellcodes
   - Ajustando a Pilha
   - Exploração Local
   - **Laboratório**
4. *Buffer Overflow* em Windows (x86 – 32 bits) – Programa 2
   - Análise de uma Aplicação Web real
   - Processo de Fuzzing do Cabeçalho HTTP com script em Python
   - Identificação de ponto de entrada vulnerável
   - Criação de uma Prova de Conceito (PoC) com Python
   - Análise de corrupção de memória da aplicação via Immunity Debugger
   - Sobrescrevendo o endereço de retorno de forma precisa via BOF
   - Identificando endereços de memória e instruções de interesse
   - Criação do *Exploit*
   - Preparação do Payload/Shellcode
   - Exploração Remota
   - **Laboratório**

PRÉ-REQUISITOS

- Conhecimentos básicos em Sistema Operacional Windows
  - Estrutura de diretórios, comandos básicos do prompt (cmd.exe), configuração de rede, verificação de serviços e processos
- Conhecimentos básicos em Sistema Operacional Linux
  - Estrutura de diretórios, comandos básicos do shell (bash), configuração de rede, verificação de serviços e processos, permissão de arquivos
- Conhecimentos básicos de Rede de Computadores, Serviços e Protocolos de Rede
- Conhecimento básico de programação/lógica de programação
- Conhecimento básico de linguagens de programação (Python e C)
- Familiaridade com a ferramenta Metasploit
- Familiaridade com ferramentas de Virtualização – VMWare e VirtualBox

PÚBLICO ALVO

- Especialistas em Segurança da Informação
- Analista de Segurança da Informação
- Pentesters
- Membros de Red Team / Blue Team
- Membros de CSIRT
- Analista de SOC
- Pesquisadores da área de Segurança da Informação
- Profissionais de TI com interesse e afinidade na área de Segurança da Informação
- Entusiastas de Segurança da Informação

MATERIAL NECESSÁRIO

- Os alunos devem utilizar suas próprias estações de trabalho (Windows, Linux ou Mac OS), com a capacidade de executar até 02 (duas) Máquinas Virtuais (VM) simultaneamente.
- Configuração mínima de 8GB de RAM, 40 GB de espaço livre em disco, porta USB e placa de rede (RJ45 ou *wireless*) para acesso à Internet.
- Desejável 02 (dois) monitores para incremento na produtividade.
- Software de Virtualização: VMWare ou VirtualBox, preferencialmente, a versão mais atualizada.

MATERIAL RECEBIDO

- As Máquinas Virtuais com as aplicações utilizadas nos exercícios
- Slides do Curso no formato PDF
- Gravação das sessões ao vivo
- Certificado de Conclusão do Curso no formato PDF (com a carga horária e ementa)
- Material ficará disponível no portal do aluno, o GoHacking Academy

CAPACIDADES ALCANÇADAS

No final do curso, o aluno estará apto a:

- Entender os princípios básicos de Arquitetura de Computadores
- Entender os princípios básicos de Gerenciamento de Memória
- Entender o funcionamento e utilização dos registradores
- Entender a organização de um binário (Linux e Windows) em memória
- Entender os fundamentos da Linguagem Assembly x86
- Entender o funcionamento da Pilha
- Compreender, escrever e utilizar Shellcode
- Utilizar Debuggers para análise de binários (Linux e Windows)
- Analisar binários de Linux e Windows em nível de linguagem Assembly x86
- Entender o processo de corrupção de memória em aplicações (Linux e Windows)
- Compreender vulnerabilidades do tipo *Buffer Overflow*
- Analisar e identificar pontos de entrada vulneráveis em binários (Linux e Windows)
- Criar E*xploits* que exploram falhas do tipo *Buffer Overflow* baseado na Pilha
- Entender os principais mecanismos de defesa contra *Buffer Overflow*
- Entender técnicas básicas de contorno de proteções

LAIOS FELIPE BARBOSA